7-21-2003

# Adaptive Live Video Streaming by Priority Drop

Jie Huang
*Oregon Graduate Institute of Science & Technology*

Charles Krasic
*Oregon Graduate Institute of Science & Technology*

Jonathan Walpole
*Oregon Graduate Institute of Science & Technology*



Citation Details

Huang, J., Krasic, C., Walpole, J., & Feng, W. C. (2003, July). Adaptive Live Video Streaming by Priority Drop. In AVSS (p. 342).

# ADAPTIVE LIVE VIDEO STREAMING BY PRIORITY DROP[?]

*Jie Huang, Charles Krasic, and Jonathan Walpole*

Oregon Health and Science University
OGI School of Science and Engineering

## ABSTRACT

In this paper we explore the use of Priority-progress streaming (PPS) for live video streaming applications with tight latency constraints. PPS is a technique for the delivery of continuous media over variable bit-rate channels. It is based on the simple idea of reordering media components within a time window into priority order before transmission. The main concern when using PPS for live video streaming is the time delay introduced by reordering. In this paper we describe how PPS can be extended to support live streaming and show that the delay inherent in the approach can be reduced to as little as 400 milliseconds while still supporting relatively fine-grain adaptation.

## 1. INTRODUCTION

Streaming video over the Internet requires us to deal with bandwidth and delay that vary over time. Many video streaming applications address this problem by adapting the quality of scalable video [25]. Scalable video encoding allows us to play a single video stream at full quality when network conditions allow, and to degrade gracefully when network resources are insufficient. However, scalable video encoding is only part of the solution for adaptive video distribution. Another critical component is an adaptive streaming mechanism for detecting the network conditions and choosing the appropriate scalable video components to send in order to achieve the best possible video quality given those network conditions.

Priority-progress streaming (PPS) is such an adaptive streaming mechanism [14] [15]. It uses a time-window-based approach in which all data packets with timestamps within a certain period of time are placed in a window and reordered into priority order before transmission. It then transmits these packets for the time duration of the window only. At the end of the window duration, it discards unsent packets and moves on to the next window. In this way, the available bandwidth is used to send the most important elements of the stream and the least important elements are dropped.

The main advantage of PPS is to support fine-grain dynamic adaptation over a variety of congestion control protocols. PPS avoids the problems of trying to estimate the target rate by monitoring the underlying transport protocol's sending rate, as is done in feedback-based schemes [2] [11] [17]. Our implementation of PPS in the Quasar video pipeline [23] [13] shows that PPS is good for streaming stored video because it never underestimates the available bandwidth, it ensures that the buffered data is not delayed longer than some threshold (determined by the window length), and it uses a buffer for priority reordering which has the benefit of smoothing video quality changes. Since PPS is a generic approach based on very simple mechanisms, it is possible to implement it with high efficiency and deploy it in network nodes.

The reordering window in PPS introduces latency, however, and this latency might be problematic for live video streaming applications. The latency characteristics of the video pipeline's streaming mechanisms partially determine the freshness of the video content. The freshness of the video content, which is measured by the end-to-end latency from a frame being captured to its

[?] This work was partially supported by DARPA/ITO under the Information Technology Expeditions, Ubiquitous Computing, Quorum, and PCES programs, NSF Grants CCR-9988440 and #EIA-0130344, and by Intel.

display, tends to be important for live video streaming applications.

Most live video streaming applications have requirements for video freshness; in other words, they have requirements for end-to-end latency. Some live video applications, such as remote surgery and remote control systems, have very stringent latency requirements and cannot run on the Internet at all. Others can run on the Internet but are difficult to deploy widely without quality of service extensions in the core of the Internet. High quality video-conferencing, which is interactive and allows a maximum latency of 200 milliseconds, is an example of an application that stresses the limits of the current Internet. Yet others such as video surveillance and live event broadcasting, which have relatively loose latency requirements, could possibly run on the current Internet with suitable choice of low-latency adaptation mechanisms. Examples of surveillance applications and live event broadcasting applications include remote child-care monitoring that allows parents to watch their children from work, automobile routing that allows the detection of congested roadways through video sensors so that drivers can reduce their travel time, environmental observation systems that help scientists monitor physical phenomena of the earth, unmanned aerial vehicles equipped with cameras for military surveillance, multicasting sport games that are unavailable through local TV channels, and on-line lectures for distance learning. These applications can tolerate latencies in the range of half a second to a few minutes.

In this paper, we explore how much of a problem the latency in PPS is for live video streaming and determine the range of video surveillance applications it can support. This paper describes how PPS can be extended to support live video streaming, and evaluates the latency implications of the approach. Our experiments show that even with fairly rudimentary scalable video technology, the latency due to adaptation in PPS can be reduced to as little as 400 milliseconds while maintaining fine-grain adaptation. This means that applications with a latency tolerance of a half second can be supported using TCP-friendly protocols on a coast to coast link in the US (where propagation delay is typically less than 100 milliseconds).

This paper is organized as follows. Section 2 introduces the basic idea of PPS and describes how it works for stored video streaming. Section 3 discusses the problems of using PPS for live video streaming. Section 4 outlines a series of experiments for evaluating the latency and adaptation granularity characteristics of PPS with different reordering window sizes, and presents results. Related work is discussed in Section 5. Finally, Section 6 concludes the paper and discusses future work.

## 2. PRIORITY-PROGRESS STREAMING FOR STORED VIDEO

PPS is a time-window-based, adaptive streaming protocol. PPS uses timestamps and priority labels to perform adaptive streaming. A window in PPS contains all data packets with timestamps within a certain period of time. The window is called an adaptation window; and the size of the adaptation window is the length of the time period, not the number of packets or number of bytes in this window. Data packets in an adaptation window are sorted into priority order before transmission. PPS then transmits these packets for the time duration of the window only; at the end of the window duration, a timer expires, unsent packets in the current window are discarded and the next window starts streaming. Dropping happens when the available bandwidth is less than the data rate.

PPS is well-suited to video streaming, but it can be used to stream any data flow that can be packetized such that each data packet can be time-stamped and prioritized.

### 2.1. Basic Streaming

Figure 1 shows an ideal example of PPS streaming with sufficient bandwidth and constant delay. Data packets with timestamps and priority labels are grouped into windows in time order. Suppose the timestamps are in milliseconds, and the window size is 100 milliseconds. Within each 100-millisecond window, packets are sorted and sent in priority order, assuming that a small number represents a high priority. Packets in a window are sent out as fast as possible. Hence, when PPS runs over TCP, it can take full advantage of TCP's burstiness. In this example, we have 100 milliseconds to send out the data in each window; and because the bandwidth is higher than the data rate, there is spare time in the 100-milisecond period. The spare time can be used for work-ahead or bandwidth skimming [13]. Streaming with insufficient bandwidth and with varying network delay is discussed in Section 2.2.

Conceptually, PPS has three components: a sender, a receiver, and a regulator, as shown in Figure 2. The regulator has a clock for the sender and a clock for the receiver. The sender clock tells the sender to start sending a new window when the time for the current window is over. The receiver clock tells the receiver when to stop receiving the current window and push whatever it has received for this window to the next stage in the pipeline. Data from this window that arrives after this point in time is considered to be too late for processing (i.e., the window has already been committed to the decoder). To

ensure that all data is not considered to be late at the receiver, there is an offset between the two clocks. Specifically, the receiver clock runs behind the sender clock to account for processing and network propagation delay. The regulator can adjust the offset to adapt to variations in the network delay, as discussed in Section 2.2.

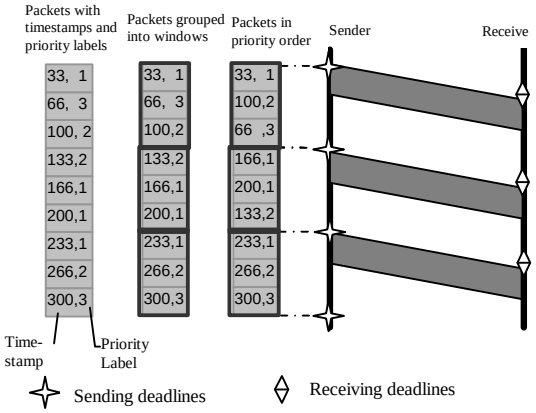


**Figure 1 PPS Streaming**

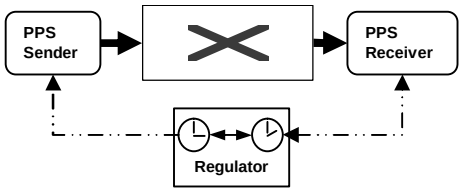


**Figure 2 PPS Modules**

The regulator can be part of the sender, part of the receiver, or a separate component. In our experiments, it resides on the receiver machine.

### 2.2. Adaptation

As shown in Figure 3, PPS can adapt to the available bandwidth. If the bandwidth is lower than the data rate, some data packets are unsent when the window time is over. These data packets, which have low priorities, are dropped. PPS makes efficient use of the limited bandwidth by transferring the data packets with highest priority first.

Figure 4 shows how PPS deals with increased delay by asking the sender to send data earlier so that it has more time to reach the receiver. When the receiver gets a packet after its window's deadline, it asks the regulator to push forward the sender clock so the offset between the sender clock and the receiver clock increases. The increased offset accommodates the larger network delay. If the delay decreases, PPS could either change back to the old sending schedule to keep the receiver buffer fill-level low, or keep the current schedule so as to prime the receiver-side buffer in anticipation of future delay and bandwidth variations.

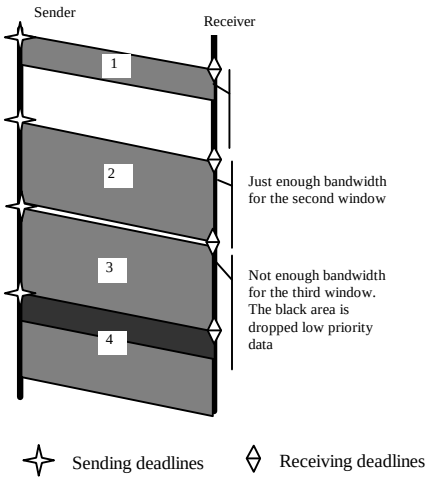


**Figure 3 Adaptation to Bandwidth**

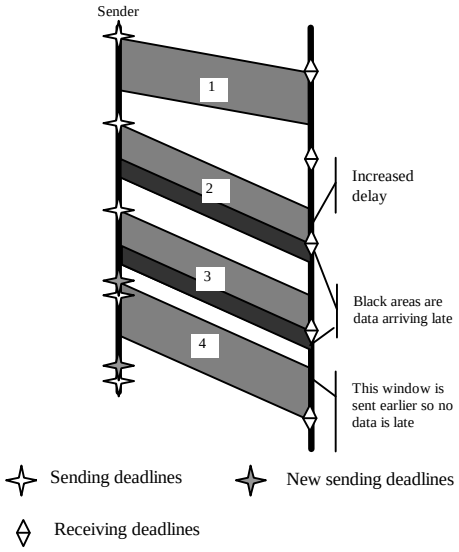


**Figure 4 Adaptation to Delay (stored video)**

In practice, bandwidth reduction and delay increase often happen together. The two adaptation mechanisms cooperate to match the varying network conditions.

### 2.3. Preparation for Video Streaming

In this section, we discuss packetization, timestamping, and prioritization for video streams.

A video stream consists of video frames; the video frames could be the data packets for PPS. However, how the video frames are encoded determines the space for adaptation. Scalable encoding is preferred because the video stream can work at different data rates and we can

achieve different quality levels under different network conditions. The Quasar pipeline uses a scalable compression format called SPEG (Scalable MPEG), extending MPEG-1 video with SNR scalability [13]. Each MPEG video frame is divided into four layers, in which the base layer contains the most significant bits of the DCT coefficients and the successive layers contain the less significant bits. Each layer of an MPEG frame is encapsulated in an SPEG packet.[1]

Video frames have inherent timestamps: the play time. SPEG packets are given the timestamps of the corresponding MPEG frames.

Prioritization exposes the video scalability to PPS in a generic way, enabling PPS to do wise adaptation without understanding the complex semantics of video encoding. In the Quasar pipeline, prioritization reflects temporal scalability and SNR scalability by reflecting dependencies among SPEG packets. For example, the base layer of an I frame has higher priority than the base layers of any P frames that depend on it, and a base layer has higher priority than the enhancement layers in the same MPEG frame. However, dependencies decide only a partial order among SPEG packets. For the importance of the base layer of a P frame over an enhancement layer of an I frame, Quasar's prioritization mechanism takes into account how much a user prefers frame rate over picture signal to noise ratio (SNR). These user preferences are provided to the Quasar pipeline in the form of utility functions [23]. To reflect the importance of an SPEG packet to the user and its importance in the video stream, Quasar's prioritization mechanism does not simply assign a priority according to the frame type and layer; instead, it uses a priority mapper [13] to calculate utility loss of the whole adaptation window to the user, assuming a particular packet is dropped. Then it decides the priority for the packet based on the utility loss. This window-based prioritization scheme results in many more priority levels for a video stream than a one-frame-based algorithm and hence supports much finer granularity adaptation. The larger the window, the more priority levels can be utilized. We combine some quality levels that are indistinguishable by human eyes and we define at most 16 priority levels at any given time in the Quasar pipeline. Details of the algorithm can be found in our earlier papers [13] [23].

SPEG is just an example scalable video format. It illustrates the effects and benefit of PPS. Other scalable compression formats such as MPEG-2 and MPEG-4 can be streamed in a similar way. We also expect that more scalability mechanisms will be available in the future, such as spatial scalability of image size, chrome scalability of color fidelities, content adaptation through addition and removal of objects, and region-of-interest. PPS can work well as long as prioritization can reflect the scalabilities.

[1] At the time we developed SPEG there was no other openly available scalable video encoding free of problematic licensing restrictions. We developed SPEG in order to exercise our adaptation mechanisms. Since this time new scalable video encoding approaches, such as MPEG-4 FGS [16], have been developed. These new codecs have better compression efficiency and finer-grain adaptation than SPEG and hence offer an even more favorable platform for PPS.

## 3. LIVE STREAMING

### 3.1. Adaptation for Live Video

Using PPS for live video introduces much more than simply replacing the stored video file with a video camera. A big difference between stored video and live video is that live video has its own capture clock. Hence, live video cannot be generated faster or slower than its capture rate, while stored video can be read whenever it is needed. This difference implies that the work-ahead mechanism described in Section 2.1 for dealing with increased delay cannot be used for live video. The capture clock also introduces the notion of end-to-end latency, which is the time from a frame being captured to its being displayed. Reducing this end-to-end latency is a goal specific to live video streaming.

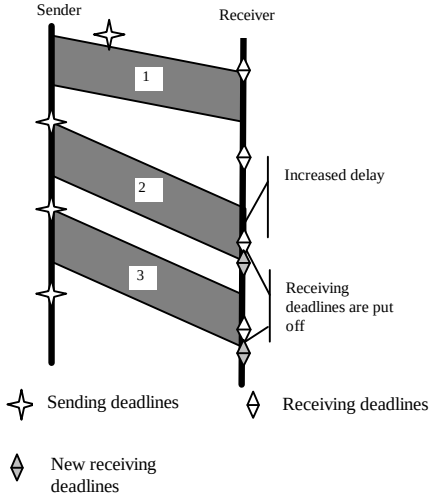


**Figure 5 Adaptation to Delay (live video)**

In the work-ahead mechanism, increases in propagation delay caused the sender to read-ahead in the video file source, and hence increase its "work-ahead"—the time difference between its clock and the receiver's clock. For live video, since it is not possible to "read ahead", we instead introduce delay at the receiver by pushing back the receiving deadlines, as shown in Figure 5. When the receiver gets a late frame, the regulator increases the offset between the sender clock and the receiver clock by pushing back the receiver clock to allow

the increased delay. In order to avoid causing burstiness in the playback, modification of the receiving deadlines is done gradually using a low-pass filter in the regulator.

Note that this mechanism is suitable for multicast because each receiver can adjust its receiving deadlines to compensate for its own network delay.

### 3.2. Latency for Streaming

The end-to-end latency from a frame being captured to its display is the main concern for live streaming. For stored video streaming, applications do require that frames arrive on time for display, but it does not matter when the frame is read from a file or how long it stays in buffers as long as it is on time for display.

There are two main sources of latency: the end machines and the network. Latency from the end machines includes the processing time and the buffering time on the sender and the receiver. On both the sender and the receiver, the processing time does not vary much. For example, the time for encoding, decoding, reordering, and prioritizing is fixed unless we improve the algorithms or switch to faster computers. Therefore we can assume that in general that these times are fixed.

Two types of buffers contribute to the total buffering time. Some buffers enable asynchrony among pipeline components. For example, the capture buffer keeps raw video frames from being dropped while the CPU is occupied by encoding or prioritizing; similarly the display buffer allows a smooth playback when a complex frame takes longer than its display duration to decode. These buffers need only be large enough to prevent the pipeline from stalling. The other type of buffer permits adaptation. The time spent in these buffers is determined by the adaptation window size, which is an adjustable PPS parameter. The latency of the network segment is something that we adapt to and cannot control.

Ignoring the latency sources that are independent of PPS, the end-to-end latency due to adaptation buffers is the sum of the adaptation window size and the transmission time, as shown in Figure 6. For PPS adaptation, the last packet in the window is delayed on the sender side for the whole window time but not delayed at all on the receiver side. Similarly, the first packet in the window is delayed on the receiver side but not at all on the sender side. All of the frames in between are delayed both on the sender and the receiver, but the total delay is always one window time. The transmission time is related to the window size in two ways. When the bandwidth is higher than the data rate, the transmission time is proportional to the amount of data in a window, which is proportional to the window size. When the bandwidth is lower than the data rate, the window size is the transmission time for this window, according to the PPS streaming algorithm (since transmission continues for the entire duration of the window before data is dropped).

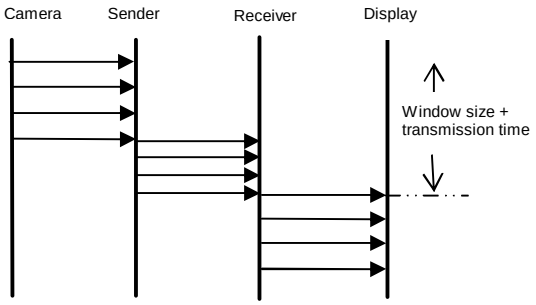


**Figure 6 End to End Latency**

In summary, for the normal case when bandwidth is limited, the latency inherent in PPS is generally twice the window size. Thus, tuning the adaptation window size is the key to tuning the end-to-end latency. For low latency streaming, a small window size is preferable. However, a small window size makes fine-grain adaptation difficult and eventually impacts video quality. This is because adaptation happens within a window, i.e. a smaller window provides fewer droppable data units and fewer priority levels for adaptation.

## 4. EXPERIMENTS

We have implemented the live Quasar pipeline by extending PPS for live streaming and substituting the MPEG source and the SPEG transcoding components of the pipeline with a camera, a capture card, and a software SPEG encoder. The capture card we use is a WinTV card from Hauppauge. The SPEG encoder is based on ffmpeg [8], an open source encoder that can encode in real time. We modified ffmpeg to implement SPEG's SNR layering strategy and to produce SPEG output directly to the live Quasar pipeline.

The live Quasar pipeline runs on Linux Mandrake 8.1. The sender and the receiver are two Pentium III 930MHz machines. The transport protocol we use is TCP. To minimize the influence of non-adjustable latency sources, we run the pipeline on a private LAN without any competing traffic. We also maintain minimum buffer fill levels that allow the pipeline to run smoothly. To feed a similar video stream for different runs, we point the camera at a static object. Thus the adaptation window size is the main control variable in the experiments.

We observe the adaptation effect through the video quality on the receiver side and through the gscope software oscilloscope [9], which is a time-sensitive visualization tool that shows the bandwidth usage, buffer

fill level, clock offset adjustment, end-to-end latency, and other useful signals in real time.

In the following subsections, we concentrate on the relationships between end-to-end latency, the adaptation window size, and the adaptation granularity. We use the adaptation granularity as an indication of the effectiveness of the adaptation. The adaptation granularity determines how closely a pipeline can utilize a given level of resource capacity, which is bandwidth in our experiments.

### 4.1. Measurement of the end-to-end Latency

We designed a novel method for measuring the real end-to-end latency including all aspects of the pipeline and I/O for capturing and displaying. We point a camera at a watch, capture a video of the watch on the sender, stream the video over the LAN, and show the watch video on a receiver; we use a second camera to take a picture of both the real watch and the displayed watch. Figure 7 is such a picture. The time difference in these two watch images is the latency of the whole pipeline. This is the most conservative way of measuring latency in the sense that it includes the latency in the camera and display hardware, which are not under our control.

**Figure 7 Measure the End to End Latency**

However, the above method is inconvenient for quantitative analysis. In the following experiments, we measure the end-to-end latency by recording the time at which a frame enters the capture buffer and the time at which it leaves for the graphic display library. The timestamp of a frame entering the capture buffer is transmitted to the receiver with the frame itself; the receiver gets the current time when sending a frame to display and compares the current time with the accompanying timestamp. The clocks on the sender and the receiver are synchronized through the Network Time Protocol (NTP).

### 4.2. Latency vs. Adaptation Window Size

Figure 8 shows the relationship between the latency and the adaptation window size. As expected, the latency grows with the window size. From Figure 8 we can see that when the adaptation window size is less than 167 milliseconds, the latency from adaptation, plus processing and necessary buffering, is well below 400 milliseconds. In the real world, the end-to-end latency also includes the network propagation delay and transmission time.

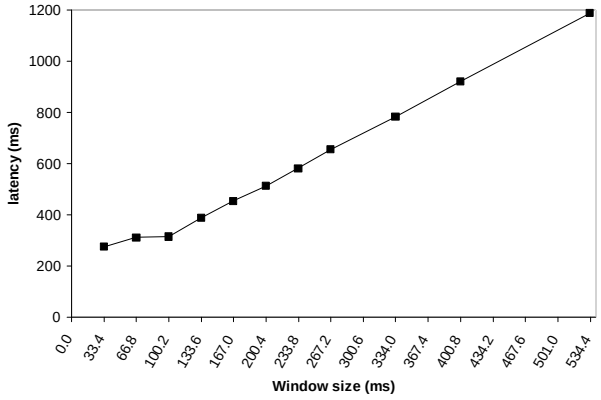


**Figure 8 Latency vs. Adaptation Window Size**

The latencies shown in Figure 8 are measured for an intra-encoded video stream. We also measure the latencies for inter-encoded video streams with GOP sizes chosen in such a way that frames in one GOP do not cross the window boundary. We find that the latencies for inter-encoded streams are very close to those for the intra-encoded stream and the GOP size has little impact on the end-to-end latency. The window size is the determinant factor for latency.

### 4.3. Latency for Streaming

Each window size can deliver a certain number of possible quality levels. These quality levels range from full quality, when all packets of the window are delivered, to zero quality when none is delivered. Between these two extremes lie a number of quality levels, one for each priority, whose bandwidth requirements can be represented as a percentage of the full quality video bandwidth. As discussed in Section 2.3, the number of priority levels and their corresponding bandwidth percentages depend on the scalability of the video stream, the window size, and the user preferences.

In Figure 9, Figure 10, and Figure 11, we show samples of quality levels available for different window sizes and user preferences. Each symbol + in the plot area represents a quality level. The *x* value of the symbol is the

window size in which that quality level is available; the *y* value of the symbol is the percentage of the full quality video bandwidth for that quality level. Figure 9 shows the available quality levels when a user prefers temporal quality and SNR quality equally; Figure 10 shows the available quality levels when a user prefers the maximum temporal quality; and Figure 11 shows the available quality levels when a user prefers the maximum SNR quality.

Ideally, for each window size there should be many available quality levels and their bandwidth percentages should be evenly distributed in order to closely match the varying network bandwidth. However, the scalability of video encoding and the window size determine how many prioritizable and independently droppable units are in a window and the sizes of these units determine the distribution of bandwidth percentage for quality levels. For the window size of 33.4 milliseconds, each window includes only one MPEG frame at NTSC rate. If no scalability is introduced, there is only one droppable unit in the window and there is only one quality level whatever the user preference is. If we double the frame rate (or double the window length), we introduce some temporal scalability and there are two MEPG frames in the window thus two droppable units and two quality levels. If we introduce SNR scalability into MPEG by using SPEG encoding and hence there are four SPEG packets in the window so there are four droppable units and at most four quality levels.

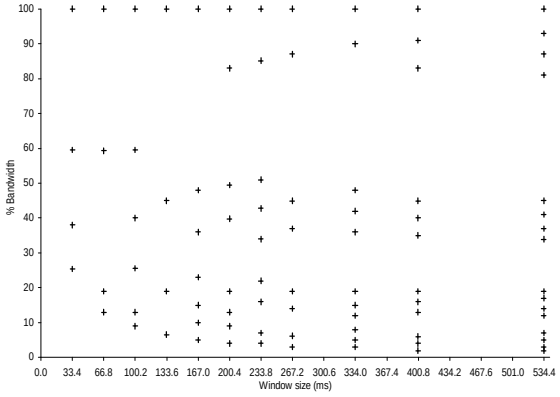


**Figure 9 Adaptation Granularity vs. Window Size (Neutral Preference)**

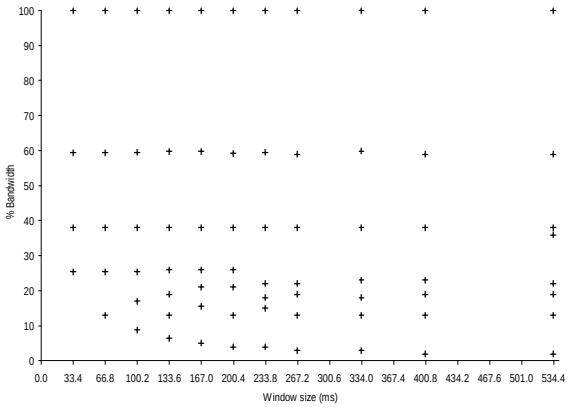


**Figure 10 Adaptation Granularity vs. Window Size (Prefer Temporal Quality)**

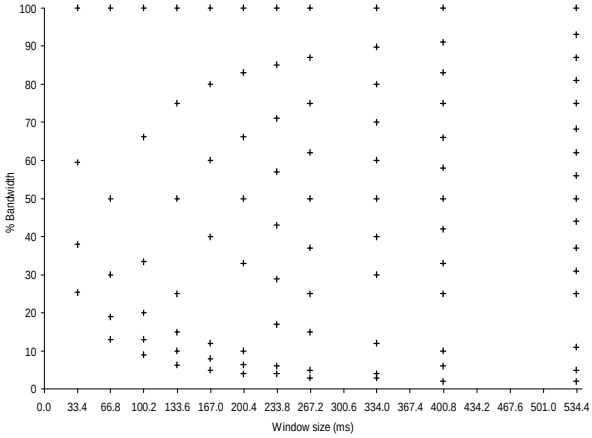


**Figure 11 Adaptation Granularity vs. Window Size (Prefer SNR Quality)**

In order to minimize latency, we need to minimize the window size while maintaining a large enough number of evenly distributed quality levels to enable fine-grain adaptation. For SPEG, a window size of 133.6 milliseconds seems to be a good choice, since it has more than 10 quality levels and allows the pipeline to achieve relatively low, less than 400 milliseconds, total end to end latency. However, SPEG has only two dimensions of quality adaptation, the temporal adaptation and the SNR adaptation; and the SNR adaptation is relatively coarse-grained. Thus, any results obtained with SPEG could easily be improved with scalable video encodings that provide finer granularity scalability. With improved scalable video encoding, PPS could easily support interactive streaming with a latency requirement of under 200 milliseconds.

## 5. RELATED WORK

Streaming video over the Internet involves many research areas: video compression, adaptive streaming, media synchronization mechanisms, application level error control, media streaming transport protocols, and so on. Wu et al. have written a comprehensive survey of video streaming approaches and directions in their paper [25]. Vandalore et al. give a detailed survey of application level adaptation techniques [24]. Our work concentrates on adaptive streaming mechanisms.

A common approach to adaptive live streaming is to monitor network conditions using feedback-based mechanisms such as RTCP receiver reports [2] and adjust video encoding parameters on the fly [17] so that the rate of the encoded video stream matches a dynamically determined target bandwidth. A key advantage of this approach is compression efficiency – the video encoder is able to optimize video quality for the given target bandwidth. Another advantage is its support for fine-grain adaptation – the target bandwidth can be chosen from a continuous range. A third advantage is low latency – adaptation can be performed without reordering data. The main disadvantages of the approach are its inability to satisfy conflicting requirements of heterogeneous receivers in a simulcast or multicast distribution network, the difficulty of tuning encoding parameters to achieve optimal video quality for a certain video rate, and the difficulty of tuning the feedback control to determine the suitable and accurate target video rate. If the target video rate is chosen incorrectly it will either result in network underutilization, congestion, or increased delay.

PPS takes an alternative approach based on scalable video encoding and priority dropping. Instead of dynamically manipulating encoding parameters, video is encoded ahead of time using a scalable encoding approach. This approach allows a wide range of scalability of the video rate, but at the expense of some compression efficiency. Adaptation is supported in PPS by prioritizing data in the scalable video stream and dynamically dropping data in priority order in order to match the target bandwidth. PPS's sending strategy does not rely on complex control models and is independent of receiver feedback. Instead, it allows an underlying congestion control protocol, such as TCP or any of the TCP-friendly streaming protocols [19] to determine the appropriate sending rate. Whatever that rate is, PPS sends video packets in priority order from a window as fast as possible. The packets that remain unsent when the time for the window is up are discarded and PPS moves to the next window. In this way, a high-bandwidth receiver gets more data than a low-bandwidth receiver for each window; but they do not interfere with the sending strategy. They both get the best possible video quality under their bandwidth limitations because for either receiver, the data packets received are more useful than those discarded, and the maximum possible bandwidth is used while preserving TCP-friendly behavior. A key advantage of this approach is the simplicity of the mechanisms and the ability to support heterogeneous simulcast distribution efficiently.

Feng et al. describe a video streaming algorithm [7], which is the closest approach to PPS described in the literature. As with PPS, it is a window-based algorithm that does priority dropping. However, in Feng's approach the window slides instead of jumping and, more significantly, the algorithm requires a priori information from stored video, which is not available for live streaming.

Miao et al. have designed an algorithm called Expected Run-time Distortion Based Scheduling (ERDBS) [18]. It is actually an adaptive streaming algorithm combined with selective retransmission, based on a sliding window. Like PPS, it decides the importance of a packet within the context of the whole window and sends the most important packets first, and it does not depend on any a priori information of the video. Unlike PPS, which considers user preferences when deciding the importance of packets, ERDBS considers the probability of packet loss and the probability that a retransmission allows a packet to arrive on time. It also recalculates the importance after each frame is sent out.

Kang and Zakhor have also designed a sliding window based reordering algorithm [12]. The sliding window size is chosen to make optimum use of the receiver's buffer. When deciding the importance of packets, they discriminate motion fields from texture fields. They assume fixed round trip time and an error-free feedback channel.

Rejaie et al. propose layered quality adaptive streaming, which adapts to varying bandwidth by adding or dropping layers [21]. They do a detailed analysis of buffer allocation to minimize latency. Their analysis is based on a rate-based congestion control mechanism called RAP rather than TCP [19]. Feamster et al [5] extend their analysis to Binomial congestion controls. Their algorithm assumes a layered-encoded stream. PPS can work with any of the TCP friendly congestion control protocols, including TCP itself, and with any of the scalable encoding schemes including layered-encoding. Unlike PPS, Rejaie et al.'s approach needs complementary error control mechanisms [6] [20].

The two main commercial video streaming systems, Windows Media [1] and Real System [3], use multi-coding for adaptive streaming. Multi-coding combines into one stream several video streams from the same content but with different bit-rates. This allows the server to switch to an appropriate stream as bandwidth changes.

PPS needs only one encoding for the same content, and hence is more efficient, enabling more scalable servers and proxy caches.

In addition to the experimental system described here, the live Quasar video pipeline is also used in project Timber [22], in which the video server is located on a robot vehicle.

## 6. CONCLUSION AND FUTURE WORK

Priority progress streaming offers a generic and efficient approach to fine-grain adaptive streaming for stored source applications. However, it implies increased latency for reordering data into priority order prior to transmission and reordering back to time order after transmission. In this paper we explored the real world impact of this reordering latency for live-source video pipelines. We showed that even for relatively coarse-grained scalable video encoding approaches reordering latency can be reduced to under 400ms, bringing the approach well within the realm of many live source video applications, such as video surveillance, and approaching the realm of interactive applications. As finer granularity video encoding approaches become available, the same level of fine-grain adaptivity will be available using even smaller reordering windows and interactive applications such as video conferencing will be easily supported using PPS.

We plan to use PPS to stream media with new types of scalability along with the temporal scalability and the SNR scalability we already support. We are also looking into approaches to reduce non-adaptation-related latency. We have implemented a real-rate scheduler, that reduces the buffer space required between pipeline components [4], and a low latency TCP that integrates congestion window and socket buffer management in order to reduce application-perceived network latency [10].

## 7. REFERENCES


[1] B. Birney. Intelligent Streaming. http://msdn.microsoft.com. October 2000.

[2] J.-C. Bolot and T. Turletti. Experience with control mechanisms for packet video in the Internet. *ACM SIGCOMM Computer Communication Review*, vol. 28, pp. 4--15, January 1998.

[3] G. Conklin, G. Greenbaum, K. Lillevold, and A. Lippman. Video Coding for Streaming Media Delivery on the Internet. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(3), March 2001.

[4] David Steere, Ashvin Goel, Joshua Gruenberg, Dylan McNamee, Calton Pu, and Jonathan Walpole. A feedback-driven proportion allocator for real-rate scheduling. In *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation*. USENIX, February 1999.

[5] N. Feamster, D. Bansal, and H. Balakrishnan. On the Interactions between Layered Quality Adaptation and Congestion Control for Streaming Video. In 11th *International Packet Video Workshop (PV2001)*, Kyongiu, Korea, April 2001.

[6] N. Feamster and H. Balakrishnan. Packet Loss Recovery for Streaming Video. 12th *International Packet Video Workshop (PV2002)*, Pittsburgh, PA, April 2002.

[7] W. Feng, M. Liu, B. Krishnaswami, and A. Prabhudev. A Priority-Based Technique for the Best-Effort Delivery of Stored Video. *In Proceedings of SPIE/IS&T Multimedia Computing and Networking*, San Jose, California, January 1999.

[8] http://ffmpeg.sourceforge.net/

[9] Ashvin Goel and Jonathan Walpole. Gscope: A Visualization Tool for Time-sensitive Software. In *Proceedings of the Freenix Track of the 2002 UNSENIX Annual Technical Conference*, June 2002.

[10] Ashvin Goel, Charles Krasic, Kang Li, and Jonathan Walpole. Supporting Low Latency TCP-based Media Streams. In *Proceedings of the Tenth International Workshop on Quality of Service (IWQoS)*, May 2002.

[11] S. Jacobs and A. Eleftheriadis. Streaming Video Using Dynamic Rate Shaping and TCP Congestion Control. *Journal of Visual Communication and Image Representation*, 9(3), 211-222, 1998.

[12] Sang H. Kang and Avideh Zakhor. Packet Scheduling Algorithm for Wireless Video Streaming. 12th *International Packet Video Workshop (PV2002)*, Pittsburgh, PA, April 2002.

[13] Charles Krasic and Jonathan Walpole. QoS Scalability for Streamed Media Delivery. *CSE Technical Report CSE-99-011*, Oregon Graduate Institute, September 1999.

[14] Charles Krasic and Jonathan. Walpole. Priority-Progress Streaming for Quality-Adaptive Multimedia. In *Proceedings of the ACM Multimedia Doctoral Symposium*, Ottawa, Canada, October 2001.

[15] Charles Krasic and Jonathan Walpole. Quality-Adaptive Media Streaming by Priority Drop. *CSE Technical Report CSE-02-015*, Oregon Graduate Institute, December 2002.

[16] Weiping Li. Overview of Fine Granularity Scalability in MPEG-4 Video Standard. *IEEE Transaction on Circuits and Systems for Video Technology*, VOL. 11, NO. 3, March 2001.

[17] H. Liu and M.E. Zarki. Adaptive source rate control for real-time wireless video transmission. *Mobile Networks and Applications*, 3:49--60, 1998.

[18] Zhourong Miao and Antonio Ortega. Expected Run-time Distortion Based Scheduling for Delivery of Scalable Media. 12th *International Packet Video Workshop (PV2002)*, Pittsburgh, PA, April 2002.

[19] R. Rejaie, M. Handley, and D. Estrin. RAP: An End-to-End Rate-based Congestion Control Mechanism for Realtime Streams in the Internet. In *Proceedings of IEEE INFOCOM*, volume 3, page 1337-1345, New York, NY, March 1999.

[20] R. Rejaie. On Design of Adaptive Internet Streaming Applications: an Architecture Perspective. *IEEE International Conference on Multimedia and Expo (ICME)*, New York, NY, July-August 2000.

[21] R. Rejaie, M. Handley, and D. Estrin. Layered Quality Adaptation for Internet Video Streaming. *IEEE Journal on Selected Areas of Communications (JSAC), Special issue on Internet QoS*, Winter 2000.

[22] http://www.cse.ogi.edu/PacSoft/projects/Timber/accomplishments.htm

[23] Jonathan Walpole, Charles Krasic, Ling Liu, David Maier, Calton Pu, Dylan McNamee, and David Steere. Quality of Service Semantics for Multimedia Database Systems. Appears in *Database Semantics: Semantic Issues in Multimedia Systems*. Edited by Robert Meersman, Zahir Tari and Scott Stevens, Kluwer Academic Publishers, January 1999.

[24] B. Vandalore, W. Feng, R.Jain, and S. Fahmy. A Survey of Application Layer Techniques for Adaptive Streaming of Multimedia. *Journal of Real Time Systems (Special Issue on Adaptive Multimedia)*. January 2000. B. Vandalore, W. Feng, R.Jain, and S. Fahmy. A Survey of Application Layer Techniques for Adaptive Streaming of Multimedia. *Journal of Real Time Systems (Special Issue on Adaptive Multimedia)*. January 2000.

[25] Dapeng Wu, Yiwei Thomas Hou, Wenwu Zhu, Ya-Qin Zhang, and Jon M. Peha. Streaming Video over the Internet: Approaches and Directions. *IEEE Transaction on Circuits and Systems for Video Technology*, VOL. 11, NO. 3, March 2001.